TABLE OF CONTENTS

chapter 1 TERMS OF ENDEARMENT
Defining Information Technology

Why Know Just the Right Word in Information Technology?
A Few Basic Terms about Hardware
A Few Basic Terms about Software
A Few Basic Terms about Ideas
Analytical Thinking

chapter 2 WHAT THE DIGERATI KNOW
Exploring the Human-Computer Interface

Learning about Technology
Learning to Use the Graphical User Interface
Basic Metaphors of Software
Standard GUI Functionality
Learning through Feedback
Learning More Advanced GUI Features
A Basic IT Principle: Process Follows Function
Searching Text Using Find
Editing Text Using Substitution
Thinking about Information Technology Abstractly

chapter 3 MAKING THE CONNECTION
The Basics of Networking

How Have Networked Computers Changed Our Lives?
Communication Types: Some Comparisons
How Networking Takes Place, or The Medium of the Message
The World Wide Web
The Internet and the Web

chapter 4 MARKING UP WITH HTML
A Hypertext Markup Language Primer

Marking Up a Web Page with HTML
Structuring Documents
Marking Links with Anchor Tags
Including Pictures with Image Tags
Handling Color in HTML
Handling Lists in HTML
Handling Tables in HTML
HTML Wrap-Up

chapter 5 SEARCHING FOR TRUTH
Locating Information on the WWW

Searching in All the Right Places
How Is Web Page Information Organized?
How Is Web Site Information Organized?

Searching the Web for Information: Search Engines
Web Page Information: Truth or Fiction?
Case Study: The Burmese Mountain Dog Page

chapter 6 SEARCHING FOR GUINEA PIG B
A Case Study in Online Research

Getting Started on Online Research: R. Buckminster Fuller Case Study
Primary Sources
Secondary Sources
Case Study Wrap-Up: Recording the Research Path

chapter 7 TO ERR IS HUMAN
An Introduction to Debugging

Precision: What Exactly Do You Mean?
Debugging: What's the Problem?
Debugging a VCR: A Five-Step Strategy
HTML Butterflies and Bugs: A Case Study in Web Page Debugging
The Printer Is Not Printing: A Classic Scenario in IT System Debugging

chapter 8 BITS AND THE "WHY" OF BYTES
Representing Information Digitally

Digitizing
Presence and Absence (PandA) of a Phenomenon
The Hex System Explained
How Text Is Digitized: Using the PandA Representation
The Oxford English Dictionary: A Case Study in Digitization

chapter 9 FOLLOWING INSTRUCTIONS
Principles of Computer Operation

Computers: Instruction Execution Engines
Performing Computer Instructions—The Fetch/Execute Cycle
Anatomy of a Computer
Integrated Circuits
How Semiconductor Technology Works
Combining the Ideas

chapter 10 A TO Z WITH AUDIO CDS
Algorithmic Thinking

Algorithm: A Familiar Idea
The Anatomy of an Algorithm: Alphabetizing CDs
Looking to the Future

chapter 11 BITS ARE IT
Representing Multimedia Digitally

Digitizing Color
Digitizing Photographs
Digitizing Sound

Digitizing a Still Web Picture or Video
Digital Representation of Virtual Reality
Bits Are It: The Bias-Free Universal Medium Principle

chapter 12 RELATIONSHIPS AND RESPONSIBILITIES
Using Computers in Polite Society

Improving the Effectiveness of Email
Creating Good Passwords
Reducing the Risk from Viruses and Worms
Protecting Intellectual Property: Copyright
Ensuring the Reliability of Software

chapter 13 TUPLES AND TABLES
Principles of Databases

"You Can Look It Up"
Defining Tables
Operations on Tables
Join the Team

chapter 14 A TABLE WITH A VIEW
Database Queries

DB Redundancy Is Bad, Very, Very, Very Bad
References and Relationships
Building a Database
Queries: Creating Views
Database Systems
ER without Trauma

chapter 15 HAI!
Adventure Case Study in Database Design

Strategy
The Problem
Needs Analysis
Table Design
The Relationships
Assessment
Implementation—Building the Database
Viewing the Database
Creating the View Queries
View Implementation
Data Entry
Lessons, Dives, and Climbs

chapter 16 eCOMMERCE
Interactive Networking for Business

The Six Big Challenges of eCommerce
The Challenge of Variation

The Challenge of the Structure of the Setting
The Challenge of Separate Events
The Challenge of Everything at Once
The Challenge of Interoperability
The Challenge of Uncertainty

chapter 17 PRIVACY AND SECURITY
When and How to Protect Information

Whose Information Is It?
Keeping Information Private
The Cookie Monster: Possible Abuses of Web Security
Encryption and Decryption

chapter 18 FOUNDATIONS OF PROGRAMMING
Concepts Expressed in JavaScript

Review and Plan of Attack
Variables, Names, and Values
Declarations
String Literals, Booleans, and Data Types
Assignment
Expressions
Conditionals
The Espresso Program
Execution for a Double Tall Latté

chapter 19 THE BEAN COUNTER
A JavaScript Program

Preliminaries
Getting Started
Creating the Interface Page
Event-based Programming
Critiquing the Bean Counter
Revising the Bean Counter
Review

chapter 20 THINKING BIG
Abstraction and Functions

Abstraction
Functions
Applying Functions
Rules for Functions
The Memory Bank
Electronic Coin Flipping
Final Touches

chapter 21 ONCE IS NOT ENOUGH
Iteration Principles

Play It Again, Sam
Range of Variation
The Principles of Iteration
Experiments with Flipping Electronic Coins
Indexing
Regularizing Array References
Animation

chapter 22 ALGORITHMIC PROBLEM SOLVING
The Smooth Motion Case Study

The Task
A Strategy for Problem Solving
Basic Structural Web Page Task
Animating the Grid Task
The Best Laid Plans . . .
Sensing the Keys Task
Detecting the Staircase Task
Overall Design Task
Finishing the Web Page Task
Assessing and Looking Back

chapter 23 COMPUTERS CAN DO ALMOST {❐ EVERYTHING, ❐ NOTHING}
Limits to Computation

Can Computers Think?
Acting Intelligently
Creativity
Universality
Practical Consequences of Universality
Macintosh versus PC
Faster and Slower
How Tough Can IT Be?

chapter 24 COMMENCEMENT
A Fluency Summary

The Ideas of IT
Fluency's Detail Spectrum
Lifelong Learning
License to Drive

chapter 1

WHAT THE DIGERATI KNOW

Exploring the Human-Computer Interface

learning objectives

> Learn to teach yourself new applications.

> Acquire the skills of experienced users.

- Use consistent interfaces.
- Expect feedback from applications.
- Don't be afraid to make mistakes.
- Try new applications.

> Understand that process follows function.

> Learn the basics of text searching, including the Placeholder Technique.

> Learn to think abstractly about technology.

PERHAPS the most uncomfortable part of being an inexperienced computer user is the suspicion that everyone but you innately knows how to use technology. They seem to know intuitively what to do in any situation. Maybe, you think, they have all come from the same alien planet that computers have.

Of course, experienced users aren't really born knowing how to use computers. They have learned through experience what to expect and how to respond in most situations. In this chapter, we reveal some secrets of the *digerati* so you, too, can "join the club."

The major goal for this chapter is to show you how to think about technology abstractly. We do this by asking how people learn technical skills and by considering what the creators of technology expect from us as users. This chapter will help you understand (1) that computer systems present consistent interfaces, standard metaphors, and common operations; (2) that computer systems always give feedback while they are working; (3) that making mistakes will not break your computer; (4) that the best way to learn new computer software is to try it out, expecting to make mistakes; and (5) that asking questions of other computer users does not mean that you are a dummy, but proves that you have an inquiring mind. These ideas can help you learn new software quickly. A key abstract idea about software is that it obeys fundamental laws of nature. This idea can help you in your everyday software usage. Finally, we demonstrate these laws using basic text searching, helping you to know how to use any vendor's software. In other words, when you know the basics, you can learn any system. You, too, can become one of the digerati.

LEARNING ABOUT TECHNOLOGY

Digerati is a new word for knowledgeable in digital technology, analogous to literati.

People know innately how to chew, cough, stand, blink, smile, and so forth. However, they do not know how to ride a bicycle, drive a car, use a food processor, or start a lawnmower. For any tool more complicated than a stick, a person needs some explanation about how it works and possibly some training in its use. Parents teach their children how to ride bicycles, drivers' training classes explain to teenagers how to drive safely, and most products come with an owner's manual.

Some tools such as portable CD players are so easy to use that most people living in a technological society find their operation "obvious." They don't need to look at the owner's manual. For example, given a CD and the player, and having seen one in use, you can guess what the controls do. (If you didn't know, the icons on the buttons, as

Figure 2.1 *Controls on a portable CD player.*

shown in Figure 2.1, would probably make little sense.) And people from technological societies can usually recover from mistakes. For example, if you insert the CD upside-down, it won't work, so you'd turn it over and try again.

But this doesn't mean that we have any innate technological abilities. Instead, the fact that you know how a CD player works without being told emphasizes two facts about technology:

- Our experience using (related) devices found in our technological society guides us in what to expect.
- The designers who create these devices know we have that experience and design their products to match what we already know.

These two facts are key to success with information technology.

LEARNING TO USE THE GRAPHICAL INTERFACE

On certain Apple Macintosh computers, when a user inserts an audio CD into the computer's CD drive, the graphical user interface (GUI) shown in Figure 2.2 appears on the screen. A GUI, pronounced "gooey," is the visual display through which users interact with computer programs. This GUI appears on the screen because the Macintosh's operating system, noting that a CD has just been inserted into the drive and recognizing that it is an audio CD, assumes the user wants to listen to it. So, it starts the software that plays audio CDs, displays the GUI to find out what the user wants to play, and waits for a response.

As first-time users of this software, we look at the GUI wondering what the software does and how to use it. There is an online user's manual, but we will not need it. The GUI tells the whole story. The GUI shows us a familiar picture of a CD player, complete with digital readout in green LCD numerals, "metallic" buttons with the standard icons, and so on. No physical CD player looks exactly like this one—for example, the CD slot is the wrong size—but it is so much like a real CD player that anyone who has seen one recognizes this picture immediately. So we guess that pressing the button with the Play icon will make the computer play the CD. But because this is a GUI, we can't really press a button. Instead, the action analogous to "pressing" is "clicking" with the mouse. We know this from experience. Clicking on the button with the icon for Play starts the CD playing. Because it worked, we know that the analogy of the

Figure 2.2 *Graphical user interface for one version of an Apple Macintosh audio CD player.*

GUI to a physical CD player is correct, and from then on we have a basic idea of how to operate the software. We didn't need lessons; no one had to help us.

Understanding the Designer's Intent

The use of the physical analogy to help the user understand how to operate the CD player software demonstrates a basic idea of consumer software. Like anyone who invents a new tool, software designers have to teach users how to operate their inventions. They write manuals explaining all of the software's slick features, but it's much better if users can figure them out without studying the manual. So, software designers, like CD player designers, try to pick easy-to-understand interfaces to their software. The designers guessed that a picture of a CD player would be the easiest way to teach the user how to use their software. They put a lot of time and effort into making their GUI look real by using LCD numerals, "metallic" buttons, the standard button icons, a "slot" for the CD (which plays no role but to support the metaphor), slider volume control, and so forth. And they guessed right. Anyone who's used a CD player will know how this software works, or at least its basics. (The Microsoft Windows operating system audio CD player for the same software generation also uses some of these features of the physical analogy, as shown in Figure 2.3.)

Amazingly, the LCD numerals even have the "shadow" characteristic of "unilluminated" segments that would make up a physical LCD digit display, though they may not be visible in the Figure 2.2 screen shot.

So it is in a software designer's interest to make the GUI easy for us to figure out on our own. Though they do not always succeed as brilliantly as the designers of the

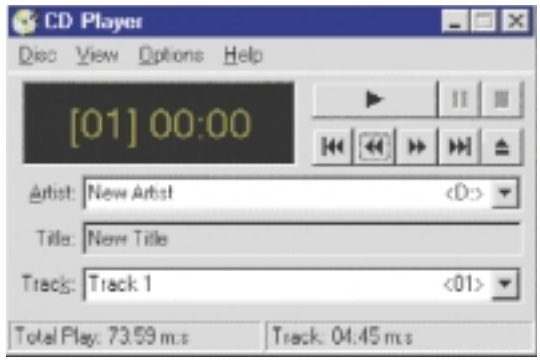


Figure 2.3 *Audio CD player GUI for the Windows operating system.*

Macintosh audio CD GUI, we should expect as users that the software has been well crafted and that we can "brain out" how it works. We use this idea every time we meet new software.

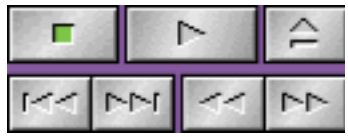
BASIC METAPHORS OF SOFTWARE

Looking again at Figure 2.2, there's more to the CD player GUI than just the seven standard buttons: Stop, Play, Eject, Last Track, Next Track, Backward, Forward. How are we supposed to know about those other features? We can guess what some of them do, like Shuffle, because we are familiar with real CD players. But we can figure out other parts of this software based on standard metaphors used in almost all consumer software GUIs. Most have graphics that look familiar, and we see them in the audio CD player. Once we learn these metaphors, we can easily guess how to interact with most software.

The following basic metaphors are universal. They are no longer unique to Apple, which introduced them in consumer software, or Microsoft, which gives them their widest distribution. They generally have a consistent meaning, though sometimes slightly different graphic forms.

Command Buttons

As shown below, command buttons usually look like a 3D rectangle, highlighted and with an icon or text centered on the button, as explained in Chapter 1. This text label names the command. To invoke the command—that is, to tell the software to perform the operation shown on the label—we are expected to "press" the button by clicking on it with the mouse. (A click is enough; it is not a good idea to press down on the mouse button for long time.) We then get feedback telling us that the button has been clicked, usually a change of color, shadow, or highlight; some text/icon change; or other indicators, including an audible "click." (Some people think such indicators are obsessive attempts at realism, but some form of feedback is essential to effective computer use, as explained below.)



Slider Controls

The volume control, (b), is a slider control. Slider controls set a value from a "continuous" range, such as volume. To move the slider, place the mouse pointer on the slider, hold down the mouse (left) button, and move in the direction of change. The most common sliders are the scroll bars in a window display (a), usually shown at the right and bottom of the window. When a window is not large enough to display all of the information in either the horizontal or vertical direction, a scroll bar is shown for each orientation in which information has been clipped. So for example, for a word processor document that doesn't all fit in a window, a scroll bar will let you move up and down or side to side, so you can read all the text. Often the size of the slider of the scroll bar is scaled to show what proportion of that dimension is displayed. Thus, if the slider takes up half of the length of the "slot," about half of the information is displayed; if the slider is a tenth of the length of the "slot," about a tenth of the informa-

(a)

(b)

tion is shown. There are usually directional triangles ( ) at one or both ends of the scroll bar; clicking on them moves the slider one "unit" in the chosen direction.

Triangle Pointers

To reduce clutter in a GUI, designers hide information until the user needs or wants to see it. A triangle indicates the availability of hidden information. Clicking on the triangle reveals the information. You can see a triangle in Figure 2.4 ( ) (below the Normal button). Clicking on the triangle pointer in Figure 2.4, for example, would result in Figure 2.5. Notice that now the direction of the pointer is reversed. Clicking on that triangle again hides the information.

Figure 2.4 *Audio CD player GUI with track numbers hidden.*

Figure 2.5 *Audio CD player GUI displaying the track numbers.*

Close Boxes

Any open window can be closed, and most GUIs give the user a way to do it with a click. On the Macintosh, clicking on the empty box ( ) in the upper-left corner closes the window. On Windows systems, clicking on the X button ( ) in the upper-right corner closes the window. A Windows application ends when its main (or only) window is closed, but if just its subwindows are closed, the application generally continues running.

These are just a few of the metaphors software designers use. There are many others, and beginning users should get to know them quickly. The point here is that computer applications have many operations in common, and software designers try to make interfaces look familiar so they can take advantage of the user's knowledge and experience. Experienced users look for metaphors, and when they find a new metaphor, they add it to their repertoire.

TIP

Mac or PC? Is the Macintosh better than the PC or vice versa? The question usually sets off a pointless argument. Some people wrongly conclude that the system they don't know must be very different and hard to use. In fact, the two systems are much more alike than they are different, sharing the concepts of this chapter and much, much more. Any competent user of one can quickly and easily learn to use the other. And *every* Fluent user should.

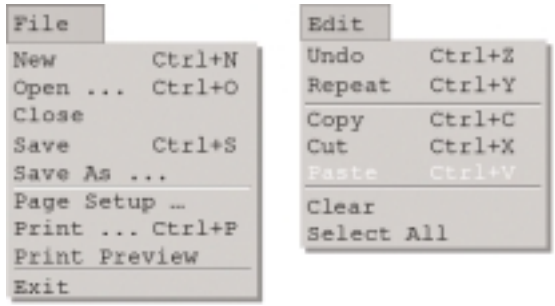


Figure 2.6 *Generic File and Edit menus.*

Menus

The main way users interface with software is through menu choices. Menus list the operations that the software can do. A menu groups operations that are similar. Menus are either listed across the top of a window, in which case they are **pull-down** or **drop-down** menus, or they appear wherever the mouse is pointing when a mouse button is clicked, in which case they are **pop-up** menus. Both menu types work the same way.

Pulling down or popping up a menu shows a list of operations. Sliding the mouse down the list causes the items to be highlighted as it passes over them, and clicking or releasing the mouse button selects a menu item. If the software has enough information, it does the operation immediately and the window closes. If not, it asks the user for more information by opening a new window. Sometimes it keeps asking for information, opening more windows, but eventually it will get enough information and the operation will be performed. At any time during this dialog you can stop simply by moving your cursor away from the menu or pressing Cancel. That is, clicking on Cancel is the same as never having looked at the menu in the first place, no matter how much information you've entered.

Menus in most consumer software give more information that just the item list. They tell you which operations are available, which ask you for more input and which have shortcuts. Look at Figure 2.6 as you read these descriptions.

Which Operations Are Available? Unlike restaurant menus that are printed and reused, occasionally requiring the server to indicate that certain items are not available, GUI menus are created each time they are opened. Because of this, they tell you exactly which operations are available. (An operation may not apply in every context. For example, Paste is not available if nothing has been Cut or Copyed.) Operations that are available are usually shown in a solid color, and operations that are not available are shown in a lighter color or "gray," as shown for the Paste operation in Figure 2.6. Unavailable items are not highlighted as the cursor passes over them, and, of course, they cannot be selected.

Is More Input Needed? Some operations need further specification or additional input. Menu items that need more information might have a triangle pointer (▶) at the right end of the entry. Selecting such an item pops up a menu with the additional choices. Making a selection causes the operation to be performed unless it requires still more selections. Some items show they need more input with an ellipsis (…) after

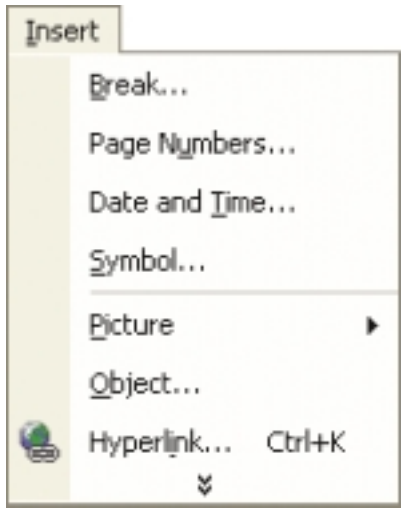


Figure 2.7 *Menu showing ellipsis and triangle pointer.*

their name. Selecting the item opens a dialog window. For example, in Figure 2.7, Break has an ellipsis because the user must give the name of the file to be opened.

Is There a Shortcut? Sometimes it is more convenient to type a keyboard character than to point and click with the mouse. A **shortcut** is a set of keyboard characters, shown next to the menu item, which when typed invokes the operation as if the menu item had been selected. For example, in Microsoft Windows, the menu choice Copy has the shortcut Ctrl C, and Paste has the shortcut Ctrl V. (Like the shift key, the control key [Ctrl] is held down while typing the associated character.) To use the copy shortcut, you would hold down the control key and type the letter "c". (Although the character is shown as a capital, you should not also press the shift key.) A control-plus-character combination is required so that the operating system can distinguish between the menu choice and a plain character. The Macintosh uses Command C and Command V for these operations—that is, the same letters. The Command key is labeled with the "clover" symbol (⌘). Notice the similarity between the two operating systems. Shortcuts are handy for people who use an application intensively. The shortcut key combination can be set or changed in most systems.

Other kinds of shortcuts for menu entries include icons, such as the printer ( ) and disk ( ) icons in a Windows toolbar.

STANDARD GUI FUNCTIONALITY

There are some operations that almost all personal computer applications do. That is, whether the information is text or spreadsheets or circuit diagrams or digitized photographs, the fact that it is information stored in a computer means that certain operations will be available. For example, the information should be printable. It should be possible to save the information to a file, open a file containing the saved information, create a new instance, and so on. You should expect to find all of these functions in almost every software application.

To help users, the standard operations are grouped—usually with other operations specific to the application—into two menus labeled File and Edit (refer back to Figure 2.6). Generally the operations under the File menu apply to whole instances of the information being processed by an application. For example, in word processing the instance is the entire document being processed, so File menu items treat a whole document. You can expect to see the following items under the File menu:

New Create a "blank" instance of the information.

Open Locate a file in permanent storage containing an instance of the information and read it in.

Close Stop processing the current instance of the information, but keep the program available to process other instances.

Save Write the current instance to permanent storage, such as the hard disk or floppy disk, using the previously given name and location.

Save As Write the current instance to permanent storage with a new name or location.

Page Setup Specify how the printing should appear on paper; changes to the setup are rare.

Print Print a copy of the current instance of the information.

Print Show the information as it will appear on the printout.

Exit or Quit End the entire application.

Notice that lines further subdivide the operations.

The Edit operations let you make changes within an instance. They often involve selection and cursor placement. The operations are performed in a standard sequence: Select-Cut/Copy-Indicate-Paste-Revise. Selection identifies the information to be moved or copied. Selection is usually done by moving the cursor to a particular position and, while holding down either the mouse button or keyboard keys, moving the cursor to a new position. All information between the two positions is selected. Highlighting, usually color reversal, identifies the selection. If the information is to be moved, it must be removed from its present position with the Cut command; otherwise, Copy saves the selected information. Next, the new location for the information is indicated, although in many applications the Indicate step is skipped and the pasted information is placed in a standard position. The Paste copies the information from temporary memory into the indicated position. Because a copy is made, the information can be pasted again and again. Often, revisions or repositioning are required to complete the editing operation. You can expect to see the following facilities under the Edit menu (refer back to Figure 2.6):

Undo Cancel the most recent editing change, returning the instance to its previous form.

Repeat Apply the most recent editing change again.

Copy Keep a copy of the selected information in temporary storage, ready for pasting.

Cut Remove the selected information and save it in temporary storage, ready for pasting.

Paste Insert the information saved by Cut or Copy; it is placed either at the cursor position or at a standard position depending on the application.

Clear Delete the selected information.

Select All Make the selection be the entire instance.

Notice that Undo is not always available because not all operations are reversible.

Figure 2.8 *Blank monthly calendar showing one month, i.e., a "blank instance."*

Because these operations are standard—available for most applications and on most operating systems—it is a good idea to learn their shortcuts, given in Table 2.1. (Clear often does not have a shortcut, to prevent accidents.) Notice that "double click"—two clicks with the (left) mouse button—often means Open.

Notice that New under the File menu creates a "blank" **instance**. What is "blank information"? To understand this fundamental idea, notice that all information is grouped into **types**, based on its properties. So, photographs—digital images—are a type of information, and among the properties of every image is its length and width in pixels. Monthly calendars are a type of information with properties such as the number of days, year, and day of the week on which the first of each month falls. Text documents are another type, and the length of a document in characters is one property. Any piece of information—image, month, or document—is an instance of its type. Your term paper is an instance of the document type of information; June 2003 is an instance of calendar type information. To store or process information of a given type, the computer sets up a structure to record all of the properties and store its content. A "new" or "blank" instance is simply the structure without any properties or content filled in. As an example, imagine a blank monthly calendar, as shown in Figure 2.8—seven columns of squares headed with the days of the week, a place to enter the month name, and so on. That's a "New" month, ready to receive its content.

File Functions		Edit Functions	
New	N	Cut	X
Open	O	Copy	C
Save	S	Paste	V
Print	P	Select All	A
Quit	Q	Undo	Z
		Redo	Y
		Find	F

Table 2.1 Standard Shortcuts. *These common shortcut letters for standard software operations combine with "Command" for Mac OS and "Control" for Windows.*

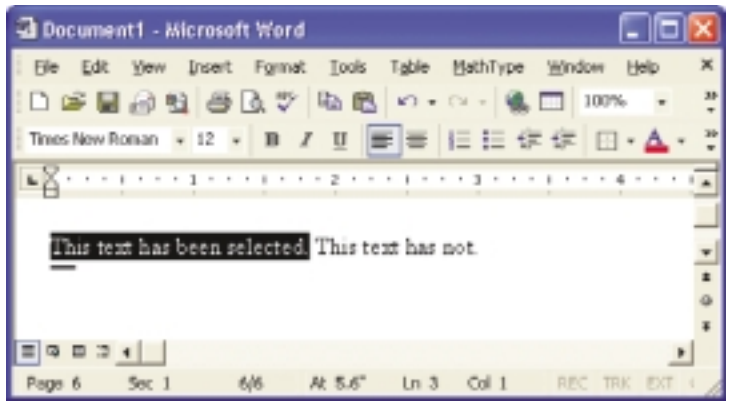

Document1 - Microsoft Word
File Edit View Insert Format Tools Table MathType Window Help
Times New Roman 12
This text has been selected. This text has not.
Page 6 Sec 1 6/6 At 5.6" Ln 3 Col 1 REC TRK EXT

Figure 2.9 *Selected text in a word-processing application.*

TIP

Be Selective. New users are often confused when a menu operation they want to use is not available, that is, it is shown in gray. Often this is because the operation requires the user to select something and nothing is selected. For example, the computer cannot perform Copy until you have told it what you want copied. See Figure 2.9 for a sample of selected and unselected text.

LEARNING THROUGH FEEDBACK

A computer is the user's assistant, ready to do whatever it is told to do. It is natural that an assistant should report back to the person who made a request and tell him or her whether the job was done. This is especially true when the assistant is a computer (and therefore not very clever) because the person needs to know that the task was done and when to give the next command. A user interface will always give the user feedback about "what's happenin'."

Feedback takes many forms, depending on what operation a user has asked for. If the operation can be performed instantaneously, that is, so fast that a person would not have to wait for it to be finished, the user interface will simply show that the operation is complete. When the operation is an editing change, the proof that it is done is that the user can see the revision. For other tasks, highlighting, shading, graying, or some other color change, or underlining tell the user that the operation is done.

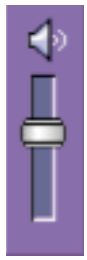
The most familiar form of feedback is the indication that the computer is continuing to perform a time-consuming operation. As the operation is being carried out, the cursor is replaced with an icon that shows elapsing time. On Windows systems, the icon is an hourglass ( ), and on Macintosh systems, it is a wristwatch with a hand advancing ( ). Applications can also give the user custom feedback. Claris software uses a circle divided into quarters, two white and two black, that "revolves." The file transfer application Fetch turns the cursor into a running dog. When the completion time can be predicted accurately, some applications show a bar that is "filled" as the operation progresses. Often these displays tell you when 100% will be reached. Finally, when an operation is processing a series of inputs, a "completion count" tells the user how many are done and how many remain.

TIP

Following Protocol. Our normal interactive use of computers alternates between our telling the computer to do something and the computer doing it. If it can't finish immediately, it gives us feedback showing the operation is in progress. If it's done, we can see the effects of the command. Be attuned to this protocol. Notice the effect the command had. If nothing seems to be happening, the computer is waiting for you to give a command.

"Clicking Around"

When the digerati encounter new software, they expect a consistent interface. They expect basic metaphors, standard operations and feedback. They automatically look for these essential features of the interface as they begin exploring. The purpose of the exploration is to learn what the software can do.



Exploring a user interface will be called **clicking around**. It involves noting the basic facilities presented by the GUI and checking each menu to see what operations are available. So, for example, the experienced user's response to seeing a slide bar is to slide it to see what happens. On the Mac audio CD GUI shown in Figure 2.2, when we slide the bar up and down, the speaker icon above it shows more-and-larger or fewer-and-smaller white arcs to its right. We guess these arcs stand for more or less sound coming from the speaker. If the CD is playing, we'll also notice that the volume increases or decreases. Either way, we know that this is the volume control.

CHECKLIST>>

Hints for a Fast Start. When you're using software for the first time:

- Take a minute—it rarely takes more time—to study the GUI's graphics.
- Open each window to see what operations are available.
- Determine the purpose of icons and controls.
- Pass the cursor over the icons and controls with the "balloon help" or "what's this?" turned on for a short explanation of what each does.

Figure 2.10 *Customized Audio CD player GUI.*

"Clicking around" is the process of figuring out what operations are available with a software application without having to read the manual or getting instructions from someone else. Software manuals can be dull reading and hard to use. But "clicking around" is not going to make them obsolete. Manuals—they're mostly online and called Help—are still necessary and useful. "Clicking around" works because (a) we come to the new software with technological experience, and (b) software designers try to build on what we know. When the new software works like the last software did, we already "know" how to use it. The manual is usually needed only to understand advanced features that may not be obvious. Ironically, the manual is most useful once we have become experienced users, not as beginners.

Returning to the audio CD GUI of Figure 2.4, when we click on the down triangle, we see the track list shown in Figure 2.5. As new users of this software, we may not immediately understand what the list is for, especially if the real CD players we are familiar with do not have a play list. But, by "clicking around," we notice either that "Track 1" can be selected like text or that when we move the cursor across the text, it changes into the "I-beam" text editing cursor. Both tell us that we can add text. Or, perhaps, we just guess that the track list doesn't need a large text box reading "Track 1" and so on, so there must be some other reason for it. No matter how "clicking around" cues us, we discover that we can edit the entries. We figure out from the editing capability that we can customize the title and songs on each track as shown in Figure 2.10.

"Clicking around" is exploration and is not guaranteed to lead us to all the features of the software. We may need to experiment and test repeatedly or give up and try again later. But we are bound to learn something. If we don't, the software product designer has undoubtedly failed to some extent.

"Blazing Away"

After familiarizing ourselves with a software application by "clicking around," the next step is to try it out. We use the term **blazing away** for trying out features assertively, perhaps without a clear idea of what they will do. "Blazing away" can be difficult for beginning users because they're afraid something will break if they make a mistake. A basic rule of information processing is: ***Nothing will break!*** If you make a mistake, the software is not going to screech and grind to a halt and then plop onto the floor with a clunk. When you make a mistake, the software may "crash," but nothing will actually break. Most of the time, nothing happens. The software catches the mistake without doing anything and displays an error message. By paying attention to these messages, you can quickly learn what's legal and what isn't. Therefore, "blazing away" can be an effective way to learn about the application even if you make mistakes.

Of course, you can still get into a mess by "blazing away." Creating a mess is often very easy. Beginners and experts do it all the time. The difference between the two is experts know another basic rule of information technology: ***When stuck, start over.*** That may mean exiting the software. It may mean rebooting the computer. It may simply mean "undoing" a series of edits and repeating them. The simple point is that the mess does not have to be straightened out because it didn't cost anything to make it in the first place, except for your time. Because that time will be chalked up to "experience" or "user training," there is no harm in throwing the mess out. Thus an experienced user who is "blazing away" on a new software system will probably exit the software and restart the application over and over, without saving anything.

TIP

Getting Out More. Starting over happens so often—it is called *getting out and getting back in*—that it has become the subject of some geek humor: A mechanical engineer, an electrical engineer, and a computer engineer are camped at Mt. Rainier. In the morning, they pack up to leave and get into their car, but it doesn't start. The ME says, "The starter motor is broken, but I can fix it," and he gets out of the car. The EE says, "No way. It's the battery, but I know what to do," and she gets out of the car. The CE says while getting out of the car, "Now, let's get back in."

Usually, we are working with new software because we want to do something in particular, so it pays to focus on getting that task done. This means that we should "blaze away" on those operations that will help us complete the task. It is not necessary to become an expert, only to complete the task. Indeed, it is common for Fluent users to know only the basics of the software systems they don't use very often. And, because they are not regular users, they often forget how the applications work and have to "click around" and "blaze away" all over again.

It is obvious that if you are "blazing away" and throwing away your efforts when you get into trouble, you shouldn't spend too much time creating complicated inputs. For example, if the software asks for text input and gives you space for several paragraphs, just enter "`Test text`" and go on exploring. Once you understand the system, you can focus on using the software productively.

LEARNING MORE ADVANCED GUI FEATURES

"Clicking around" and "blazing away" are the first steps when learning new software because all you need are your own observation and reasoning skills. If you need to know something very specific about the software, you can always read the manual or online help. However, complicated software systems usually have some features that are too advanced or specialized to discover on your own by clicking around, but that you would not think to look up. They are GUI features most of us don't even know we need.

The Shift-Select Operation

An example of this kind of feature is the use of the shift key in selection operations. Suppose we want to select the red and green circles of the stoplight in Figure 2.11a so that we can change their color, but not the yellow circle. Clicking on the red circle selects it (Figure 2.11b), as shown by the small boxes around the circle. Clicking on the green circle selects it and deselects the red circle (Figure 2.11c). Dragging the cursor across the red to the green selects all the circles (Figure 2.11d). But how do we select just red and green without the yellow? The problem is that when we select something (e.g., the green circle), anything that is already selected (e.g., the red circle) becomes deselected automatically. We need some way to

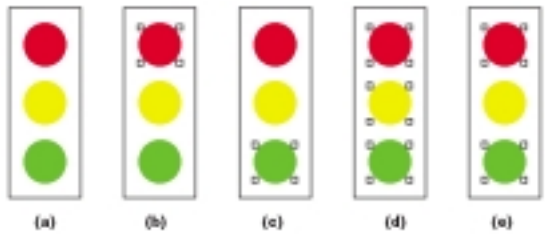


Figure 2.11 *Examples of selection.*

get around that automatic protocol.

The solution is to select the first item (e.g., the red circle) and then hold down the shift key while selecting the second item (e.g., the green circle). Using the shift key during a selection means "continue selecting everything that is already selected." Because the red circle is already selected when the green is clicked-with-shift, both become selected, which is what we wanted to do.

Notice that for text, shift-select usually results in the selection of all of the text between the cursor's previous position and its new position; it is not usually possible to select disconnected sequences of text.

The click-with-shift or shift-select operation, meaning "continue to select the item(s) already selected," is a common feature in commercial software, although the exact meaning differs with different applications. Without knowing about shift-select, however, we probably wouldn't discover it by "clicking around" or "blazing away." We would not think to try it. We might not even know that we need the feature in the first place. So how do we learn about this kind of feature?

We could take a course on the specific software or read the whole user's manual. But perhaps the best thing to do is to watch other users when they are using the software. As you watch someone using an application that you are also familiar with, you should be able to follow what he or she is doing, though it might seem very fast. If you see an operation that you do not understand, ask what the person did. Rather than thinking that you are a dummy, most people will be eager to show off their know-how. Many an obscure feature, trick, or shortcut is learned while looking over the shoulder of an experienced user, so it pays to pay attention.

TIP

Toggling shift-select. Generally when you use shift-select, one or more additional items will be selected because the click is applied to an unselected item. But what happens when you use shift-select on an item that is already selected? It deselects that item only, leaving all other items selected. This property of changing to the opposite state—selecting if not selected, deselecting if selected—is called **toggling**. It is a handy feature in many situations.

A BASIC IT PRINCIPLE: PROCESS FOLLOWS FUNCTION

A theme of this chapter is that computer systems are very similar because the designers of software want users to figure out on their own how a system works based on prior experience. To help this self-instruction process, designers use consistent interfaces and suggestive metaphors. Designers could be extremely creative thinking up wild new interfaces and unusual operations. Such GUIs might be quite interesting and very cool, rather like video games with a practical purpose, but it might take years to learn and be effective with such tools. Few of us can take that much time to learn. So instead, designers make use of the fact that consistency and familiarity help users learn quickly.

But a much deeper principle is also at work here. Designers are not just using good sense developing software that behaves in a consistent way. They are also obeying laws of nature. Logic puts limits on what information can be recorded or what operations can be computed. The laws governing information and computation are too complicated for this textbook, but they are just as fundamental as the laws that govern the structure of the atom.

These laws tell us an important fact about information technology: The task—*not* the specific software implementation—dictates the behavior of a solution. We should expect different software implementations for a task to be similar, not only because designers want them to be easy to learn, but also because they perform the same basic functions. We describe this property by the design maxim **form follows function**.

When we say "form follows function" in software, we do not mean that the systems look alike. Application software from different makers can look and feel very different even though it is for the same task. The **form** we are talking about is the way the basic operations of the software work.

Similar Applications Have Similar Features

So, for example, word processors all do the same sorts of things in similar ways, no matter which software company created them. The differences (and there are always differences, often big differences) are limited to the look, feel, and convenience of the software; the core functions are still the same. Word Perfect, Word, NotePad, Apple Works, Simple Text, BBText, and a dozen other systems give you a basic set of operations on text characters. They let you move a cursor around the text to select text characters and to create, copy, insert, and delete characters. They let you create an "empty" file of text—systems use the term new—as well as save it, name it, display it, and print it. All of those features are fundamental to text processing—they were not invented by the software companies.

The same thing applies to browser programs, spreadsheet programs, drawing programs, and so on. When we learn to use an application from one software maker, we learn the core operations for that task and the features and quirks of that vendor's product. When we use software from a different vendor, we should look for and expect to recognize immediately the same basic operations. They may have a different look and feel in the second vendor's software, but they will still be there.

Taking Advantage of Similarities

Recognizing common, basic operations in programs that do the same task frees us in at least three ways.

- When a new version of software is released, we should expect to learn it very quickly because it will share the core functions and many of the quirks of the earlier version.
- When we get another vendor's software for an application we know well, we should expect to use (its basic features) immediately.
- When we are frustrated by one vendor's software, we should try another vendor's software. Using our experience with the first system, we will learn the new system quickly. (And "voting" by buying better software should help improve overall software quality.)

In summary, because the function controls how a system works, different software implementations must share basic characteristics. You don't need to feel tied to a particular software system that you learned years ago. You should experiment with new systems; you already know the basic functional behavior.

SEARCHING TEXT USING FIND

The idea that form follows function has another advantage: It lets us learn how to use software without referring to any specific software system. Of course, we must focus only on the fundamental processing behavior rather than on the "bells and whistles" of the GUI, but learning in this way lets us apply what we know to any implementation. We demonstrate this kind of learning with text searching.

Many applications let us search text. Often called **find**, text searching is found in word processors, browsers (to look through the text of the current page), email readers, operating systems, and so on. Find is typically available under the Edit menu because locating text is often the first step in editing it. In cases where editing doesn't make sense—say, when looking through a file structure in an operating system—find may be listed under the File menu or as a "top level" application. The shortcut for find—Command F for Mac OS and Ctrl F for Windows—is standard with most applications.

The things to be searched for are called **tokens**. Most often the tokens are simply the letters, numbers, and special symbols like @ and & from the keyboard, which are called **characters**. However, sometimes we search for composite items, such as dates, that we want to treat as a whole. In such cases, the date would be the token, not its letters and digits. For the purposes of searching, tokens form a sequence, called the **text**, and the tokens to be found are called the **search string**. One property of the search string is that it can be made of any tokens that could be in the text. That is, if the text can contain unprintable characters like tab, the search string must be allowed to have those characters.

How Is a Basic Text Search Done?

To illustrate searching, suppose the search string is "`content`" and the text is a sentence from Martin Luther King's "I Have a Dream" speech:

```
I have a dream that my four little children will one day
live in a nation where they will not be judged by the color
of their skin, but by the content of their character.
```

Searching begins at the beginning or the current cursor position if there is a cursor. Though computers use many clever ways to search text, the easiest one to understand is to think of "sliding" the search string along the text, comparing at each position to see if there is a token match. This simply means looking at corresponding token pairs to see if they are the same.

```
I have a dream . . .
content
```

(Notice that spaces are characters, too.) If there is a match, then the process stops and you are shown the found instance. But if there is no match, slide the search string one position along and repeat.

```
. . . by the content of . . .
. . . ccccccccontent
```

If the search string is not found when the end of the text is reached, the search stops and is unsuccessful. (Search facilities typically give you the option to continue searching from the beginning of the text if the search did not start there.) The search ends where it began when the search string is not found.

Character searching is easy, but to be completely successful, we must be operationally attuned. One complication is that the characters stored in a computer are case sensitive, meaning that the uppercase letters, such as *R*, and lowercase letters, *r*, are different. So a match occurs only when the letters *and* the case are identical. A case-sensitive search for "`unalienable rights`" fails on Jefferson's most famous sentence from the *Declaration of Independence*:

```
We hold these truths to be self-evident, that all men are
created equal, that they are endowed by their Creator with
certain unalienable Rights, that among these are Life, Liberty
and the pursuit of Happiness.
```

To find "`unalienable rights`" in a text that uses the original capitalization, we would have to ignore the case. Search facilities will be case sensitive if case is important to the application. For example, word processors are usually case sensitive, but operating systems may not be. If the search is case sensitive, the user will have the option to ignore case.

Format Tags. Characters are stored in the computer as one continuous sequence. The characters are of two types: the keyboard characters that we type, and possibly formatting information added by the software application. Because every system uses a different method for the formatting information and because it is usually not important to the search anyhow, we will show the formatting information here using our own invented tags. *Tags* are abbreviations in paired angle brackets, such as `<Ital>`, that describe additional information about the characters—in this case, that they should be printed in italics. Tags generally come in pairs so that they can enclose text like parentheses. The second of the pair is like the first, except with a slash (`/`) or backslash (`\`) in it. For example, to show that the word "Enter" should be printed in italics, a software application might represent it as `<Ital>Enter<\Ital>`. The application won't show these formatting tags to the user, but they are there.

For example, the balcony scene from Shakespeare's *Romeo and Juliet* appears in print as:

SCENE II. *Capulet's orchard.*

Enter Romeo.

Romeo. He jests at scars that never felt a wound.

[*Juliet appears above at a window.*

But, soft! what light through yonder window breaks?

It is the east, and Juliet is the sun.

But it might be stored in the computer as:

```
SCENE·II.·⧫<Ital>Capulet's·orchard.<\Ital>↵↵<Center><Ital>Enter<\
Ital>Romeo.<\Center>↵↵<Ital>Romeo.<\Ital>⧫He·jests·at·scars·that·
never·felt·a·wound.↵<Right>[<Ital>Juliet·appears·above·at·a·windo
w.<\Ital><\Right>↵But,··soft·!·what·light·through·yonder·window·b
reaks?·↵It·is·the·east,·and·Juliet·is·the·sun.↵
```

Tags will be needed often in our study, so backslash (\) will be used here for the tags of our generic application to distinguish them from later uses in HTML, the OED digitization, and XML.

The word processor's tags surround the italic text (**<Ital>**, **<\Ital>**) and the text to be centered (**<Center>**, **<\Center>**) or right justified (**<Right>**, **<\Right>**). The user typed the other characters, and they are the ones we are interested in now. These characters include the text we see as well as formatting characters we can't see: spaces (·), tabs (⧫), and new lines (↵). Because these characters control formatting and have no printable form, there is no standard for how they are displayed; for example, new-line is the paragraph symbol (¶) in some systems. Users can ask that all the characters they type be displayed:

```
SCENE·II.·⧫    Capulet's·orchard.↵
↵
                          Enter·Romeo.↵
↵
Romeo.⧫  He·jests·at·scars·that·never·felt·a·wound.↵
                                  [Juliet·appears·above·at·a·window.↵
But,··soft·!·what·light·through·yonder·window·breaks?↵
It·is·the·east,·and·Juliet·is·the·sun.↵
```

Because the effects of the formatting are shown, it is easy to see where the nonprinted formatting characters are. During a search, the software's formatting tags are generally ignored, but all of the characters typed by the user are considered. Some systems do allow tags to be searched by giving you a way to search for formatted text, such as italic.

More Techniques of Searching

It gets more complicated when we think of search strings as having a meaning more complex than tokens. For example, we often look for words, though the tokens are characters. The problem is that the software searches for token sequences, not the more complicated objects that we may have in mind. For example, searches for the search string "**you**" in President John Kennedy's inaugural address turn up five hits:

```
And so my follow Americans: Ask not what
your country will do for you—ask what you
can do for  your country.

My fellow citizens of the world: Ask not
what America will do for you, but what
together we can do for the freedom of man.
```

Of the five hits, only three are the actual word we're looking for; the other two hits *contain* the search string. To avoid finding "`your`", we could search for "`·you·`" because words in text are usually surrounded by spaces. However, that search discovers *no* hits in this quote because "you" doesn't appear with spaces on both sides. The five hits for "`you`" are followed by "`r`", a hyphen, a new-line, an "`r`", and a comma, respectively. The "`you`" at the end of the second line probably should have had a space between it and new-line, but the typist left it out. Because looking only for the word "`you`" and avoiding "`your`" would mean checking for all of the possible starting and ending punctuation characters as well as blank, it is probably better to give up on finding the exact word matches, and simply ignore the cases where the search string is part of another word. If the search is part of the system where "words are primitive," such as a word processor, the ability to search for words will be available. Such cases are the same as changing the tokens from characters to words.

A similar problem happens with multiword search strings. The words of a multiword string are separated by spaces, but if the number of spaces in the search string is different from the number in the text being searched, no match will be found. For example, the search string "`That's·one·small·step·for·a·man`" will not be found in the quote

```
That's·one·small·step·for·a··man,·one·giant·leap·for·mankind.
```

from Neil Armstrong's words on stepping onto the surface of the moon because there are two spaces between "`a`" and "`man`" in the text. We could be careful about separating words by only one space when we type, but often we do not type the text we search. For that reason it is a good idea to avoid multiword searches and look instead for single words that apply to the context. For example, looking for "`leap`" or "`mankind`" might work because they were probably not used again while Armstrong was on the moon.

In summary, searching is the process of locating a sequence of tokens, the search string, or in a longer sequence of tokens, the text. Character searches are usually limited to the characters the user has typed, though other characters may be there. User-typed characters can include nonprintable formatting characters like new-line. Searches look for token sequences, and the tokens—for example, characters—are often more primitive than what we can build from them—for example, words. To be successful, we must think up search strings so that we find all the matches we're interested in.

EDITING TEXT USING SUBSTITUTION

Search-and-replace, also known as **substitution**, is a combination of searching and editing to make corrections in documents. The string replacing the search string is called the **replacement string**. Though substitution can apply to only one occurrence of the search string in the text, there is little advantage of search-and-replace over simply searching and editing the occurrence directly. The real power of substitution comes from applying it to all occurrences of the search string. For example, if you typed "west coast" in your term paper but forgot that regions are usually capitalized, it is a simple matter to search for all occurrences of "`west coast`" and replace them with "`West Coast`".

Because substitution can be a powerful tool that we want to study closely, we will express it in this book using a left pointing arrow (←) between the search string and the replacement string. The capitalization example is shown as

```
west coast ← West Coast
```

Such an expression can be read, "west coast *is replaced by* West Coast" or "West Coast *substitutes for* west coast." Another example is

```
Norma Jeane Mortensen ← Marilyn Monroe
```

describing her 1946 name change when she signed her first movie contract.

We emphasize that the arrow is only a **notation** that helps us discuss substitutions in this book. When you are using an application, a GUI will specify the replacement. For example, look at Figure 2.12. The two text windows of the GUI correspond to the information on each side of the arrow. Find is the left side of the arrow, and Replace is the right side. We don't type the arrow in applications. It is only for our use here.

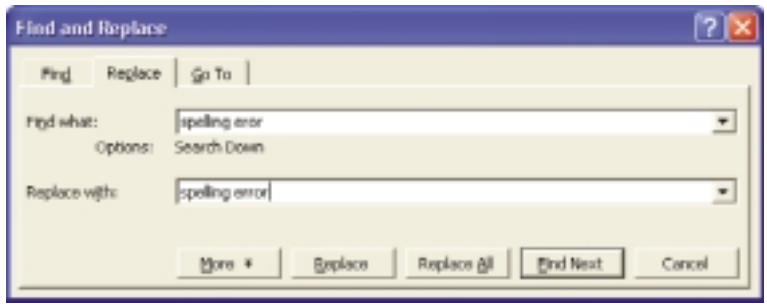


Figure 2.12 *A word processor's Find and Replace window..*

Unwanted Spaces

In the last section, we noted that multiple spaces separating words in a text complicates searching for multiword strings. Substitution can fix the "multiple spaces in a document problem": Simply collapse double spaces to single spaces. That is, if the search string is "••" and the replacement string is "•", a search-and-replace over the whole document results in all pairs of spaces becoming single spaces. Expressed using the arrow notation, the "two spaces are replaced by one " substitution is

```
•• ← •
```

Of course, in some places, such as at the end of sentence, we might want double spaces. Such cases can be fixed with substitutions of the form

```
.• ← .••
?• ← ?••
!• ← !••
```

which will restore the sentence-ending double blanks after the three punctuation characters. Performing multiple changes on text is a valuable technique.

Multiple Changes

One situation where substitution is particularly handy is when text is imported into a document from another source and the formatting becomes messed up. For example, you find the Articles from the UN's Universal Declaration of Human Rights on the Web:

Article 1 All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

Article 2 Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, color, sex, language, religion, political, or other opinion, national or social origin, property, birth or other status.

Furthermore, no distinction shall be made on the basis of political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing, or under any other limitation of sovereignty.

Article 3 Everyone has the right to life, liberty and security of person.

But when you copy the first three articles and paste them into your document, they come out looking this way:

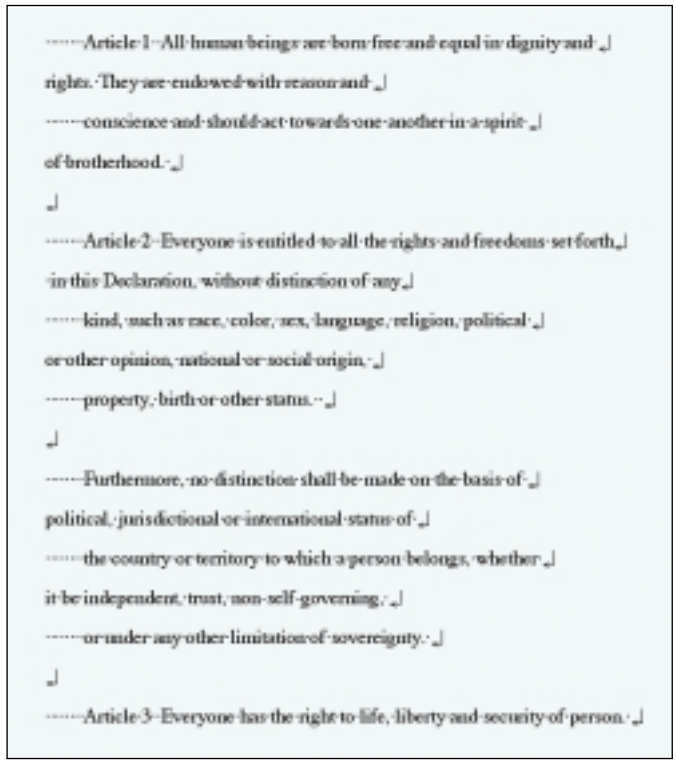
Article 1 All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood.

Article 2 Everyone is entitled to all the rights and freedoms set forth in this Declaration, without distinction of any kind, such as race, color, sex, language, religion, political or other opinion, national or social origin, property, birth or other status.

Furthermore, no distinction shall be made on the basis of political, jurisdictional or international status of the country or territory to which a person belongs, whether it be independent, trust, non-self-governing, or under any other limitation of sovereignty.

Article 3 Everyone has the right to life, liberty and security of person.

The formatting is a mess. Displaying the text with the formatting characters reveals:



We see that extra spaces and new-line characters have been inserted when we imported the text into the document.

Clearly, removing the groups of eight leading blanks is simple: Replace them with nothing. When writing the substitution expression, we express "nothing" with the Greek letter epsilon, which is called the **empty string**—that is, the string with no letters.

Notice that epsilon is used only for writing out substitution expressions for ourselves. In the find-and-replace of an application, as in Figure 2.12, simply leave the replacement string empty.

$$\cdot\cdot\cdot\cdot\cdot\cdot\cdot\cdot \leftarrow \varepsilon$$

Removing the leading blanks was easy because they are only at the beginning of the lines and nowhere else. Correcting the new-line characters is more of a problem. We want to get rid of the new-lines that have been inserted within a paragraph and keep the paired new-lines that separate the paragraphs. But getting rid of single new-lines

$$\text{↵} \leftarrow \varepsilon$$

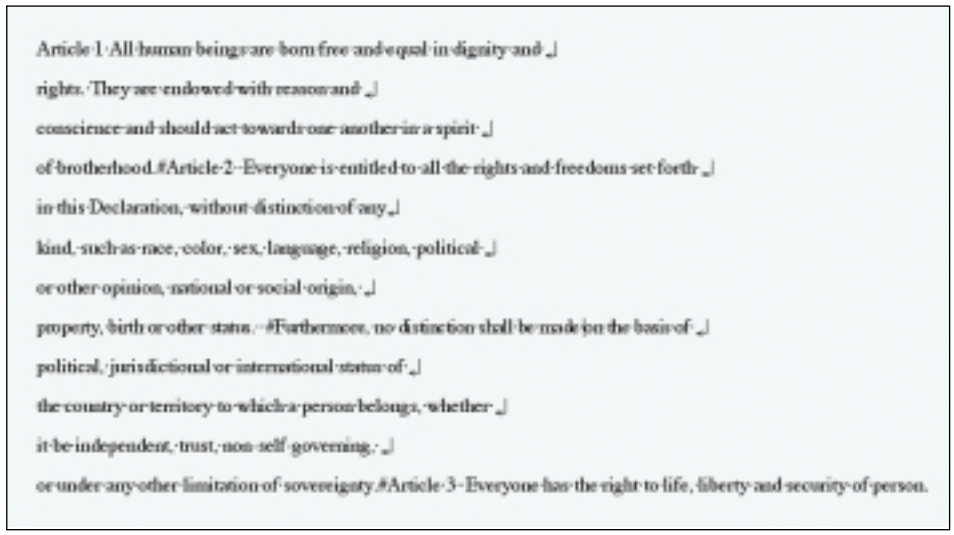
will also get rid of all the new-lines! How can we keep the paired new-lines but remove the singles?

The Placeholder Technique

An easy strategy, called the **Placeholder Technique**, solves such problems. It begins by substituting a placeholder character for the strings we want to keep, that is, the new-line pairs. We pick # as the placeholder because it doesn't appear anywhere else in the document, but any unused character or character string will work. The substitution expression is

↵↵ ← #

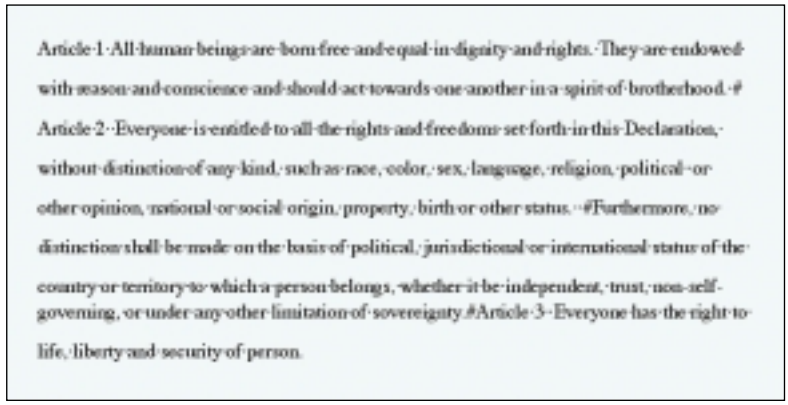
Our text without the leading blanks and double new-lines now looks like this:



The new-lines that remain are the ones to be removed, so we need to replace them with nothing

↵ ← ε

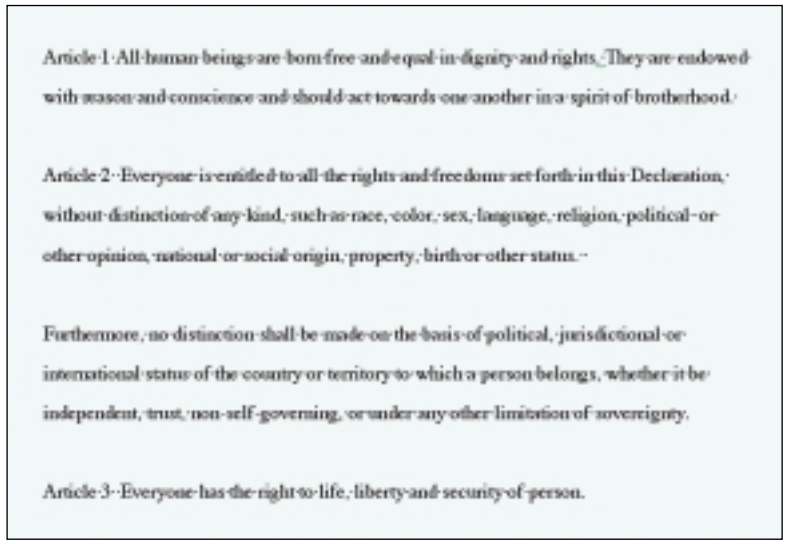
The resulting text has no new-line characters left:



Finally, replace the placeholder with the desired character string

```
# ← ↵↵
```

which gives us:



Article·1·All·human·beings·are·born·free·and·equal·in·dignity·and·rights.·They·are·endowed·with·reason·and·conscience·and·should·act·towards·one·another·in·a·spirit·of·brotherhood.·

Article·2··Everyone·is·entitled·to·all·the·rights·and·freedoms·set·forth·in·this·Declaration,·without·distinction·of·any·kind,·such·as·race,·color,·sex,·language,·religion,·political·or·other·opinion,·national·or·social·origin,·property,·birth·or·other·status.··

Furthermore,·no·distinction·shall·be·made·on·the·basis·of·political,·jurisdictional·or·international·status·of·the·country·or·territory·to·which·a·person·belongs,·whether·it·be·independent,·trust,·non-self-governing,·or·under·any·other·limitation·of·sovereignty.

Article·3··Everyone·has·the·right·to·life,·liberty·and·security·of·person.

Except for the bold form of **Article**, the result looks like the original document with only new-line pairs and no singletons. One final replacement

```
Article ← Article
```

completes the task.

To summarize, the Placeholder Technique is used to remove short search strings that are part of longer strings that we want to keep. If we removed the short strings directly, we'd trash the longer strings. The idea is to change the longer strings into the placeholder temporarily. Of course, a single placeholder character can replace the long strings because all we're keeping track of is where the longer string is located. With the longer strings replaced by the placeholder, it is safe to remove the short strings. Once they are gone, the longer string can replace the placeholder. The substitution expressions,

```
LongStringsContainingInstance(s)OfAShortString ← Placeholder
ShortString ← ε
Placeholder ← LongStringsContainingInstance(s)OfAShortString
```

summarize the idea.

BITS&BYTES

How Often? No rule says when to back up. Organizations must follow a fixed schedule, but individuals can simply assess the risk of losing everything. Hard use makes laptops more likely to fail than desktops; very new and very old equipment is more likely to fail than a middle-aged system. There is always risk.

THINKING ABOUT INFORMATION TECHNOLOGY ABSTRACTLY

We began this chapter promising to reveal some secrets known to expert computer users. And we have. Now, it is not so miraculous that the digerati appear to know how to use software they have never seen before.

Then we observed that application software systems must behave in ways governed by the functions they provide. Form follows function was our description of it. So, creating and editing keyboard input requires a small set of basic operations that all editing and word processing systems must have. The same applies to browsers, spreadsheets, and so forth. This means that when we learn specific software for a specific task, we are learning both its core operations, common to all software for that application, as well as the "bells and whistles" of its GUI. So, once we've learned one vendor's software for a task, we should expect to be able to use another's for the same task without much difficulty. Our introduction to the core ideas of searching and substitution illustrated the point: We learned the basics without needing to look at any specific software. In addition, we learned some useful skills, for example, the **Placeholder Technique**.

But the chapter's topic really concerned information technology more abstractly. We considered how people learn technology generally, and information technology in particular. Because no one is born knowing how to use technology, users must learn each new tool. The best case is when the training is simply a user's previous experience with technology. In such cases, the technology operates exactly the way users expect. Software designers try for this by using familiar interfaces, consistency, standard metaphors, standard operations, and so on. So one way to explore new software is to "click around," learning it by applying what we already know, and by not being afraid to make mistakes. Thinking from the abstract to the specific guided us to using technology well. The larger lesson of this chapter, then, is to think about information technology abstractly.

Looking Ahead

CHECKLIST>>

Thus, as members of the digerati, we will think about technology abstractly, and we're likely to ask such questions as:

- ☑ "What do I have to learn about this software to do my task?"
- ☑ "What does the designer of this software expect me to know?"
- ☑ "What does the designer expect me to do?"
- ☑ "What metaphor is the software showing me?"
- ☑ "What additional information does this software need to do its task?"
- ☑ "Have I seen these operations in other software?"

CHECKLIST>> When we think about information technology in terms of our personal or workplace needs, we may ask questions such as:

- "Is there information technology that I am not now using that could help me with my task?"
- "Am I more or less productive using this technological solution for my task?"
- "Can I customize the technology I'm using to make myself more productive?"
- "Have I assessed my uses of information technology recently?"

These and similar questions can help us use technology more effectively in our work and in our personal lives. Information technology, being a means rather than an end, should be continually assessed to assure that it is fulfilling those needs as they and the technology change and evolve.

SUMMARY

This chapter began by asking you to think about how people learn to use technology. We concluded that people must either be taught technology or figure it out on their own. We can figure out software because designers use consistent interfaces, metaphors, standard functionality, and so on. We admired how the "perfect GUI" was perfectly intuitive, allowing us to apply our previous experience to learn new applications, just like the digerati. We learned that in computer software, nothing will break when we make mistakes, so we should explore a new application by "clicking around." We should also try it out by "blazing away," knowing that we will mess up; when we do, we will throw away our work by exiting and starting over—*getting out and getting back in*.

Exploration is not the only way to learn, however. Some functions of a new application, like the shift-select, are not obvious, so we should watch other users and ask questions. You're not a dummy if you ask, only if you don't. We also learned that *form follows function*, so although software systems for a given task might look different, there are basic operations that they must have in common. If we look past the "flash" to these basic operations, we can easily learn to use another vendor's software for the task. To demonstrate this, we studied searching and substitution, which are available with many applications. And we learned the Placeholder Technique for editing text. Finally, we discussed thinking abstractly about technology. We must continually consider whether the technology we are using is helping us, whether we can be more productive with the technology we use, and whether we need to expand our uses of technology.

great moments

"MILE RUN"

When Moroccan miler Hicham El Guerrouj broke the world record on July 7, 1999, news reports trumpeted that he "smashed," "eclipsed," and "shattered" the world's record set six years earlier by Noureddine Moreceli of Algeria. El Guerrouj had run a mile in an astonishing 3 minutes, 43.13 seconds, an impressive 1.26 seconds faster than Moreceli. The descriptions were not hyperbole. People around the world truly marveled at El Guerrouj's accomplishment, even though 1.26 seconds seems like a small amount of time.

To put El Guerrouj's run into perspective, notice that 45 years had passed since Englishman Roger Bannister attracted world attention as the first man in recorded history to run a mile in less than 4 minutes. His time was 3:59.4. In 45 years, the world's best runners improved the time for the mile by an astonishing 16.27 seconds. (Notice that El Guerrouj's 1.26 seconds was a big part of that.) As a rate, 16.27 seconds represents an improvement from 15.038 miles per hour to 16.134 miles per hour, or just over 7%. Given that Bannister's world-class time was the starting point, an improvement in human performance of that size is truly something to be admired.

How do these world champions compare to average people? Most healthy people in their early 20s—the age group of the world-record setters—can run a mile in 7.5 minutes. This number includes a majority of the people in the age range, and it is approximately twice the time El Guerrouj needed. To say El Guerrouj is twice as fast as an average person is to say he is faster by a factor-of-2. This factor-of-2 difference is a rough rule for the performance gap between an average person and a world champion for most physical-strength activities such as running, swimming, jumping, and pole vaulting. The factor-of-2 rule tells us that no matter how hard most people try at physical activities, their performance can improve by at most twice. Of course, most of us can only dream of that factor-of-2 potential. Nevertheless, the factor-of-2 human standard is an important benchmark.

When we compared world champions, we said there was a 7% improvement and that El Guerrouj's speed was about a factor-of-2 times faster than the speed of an average person. There is a difference between expressing improvement as a percentage and expressing improvement as a factor. We find a factor of improvement by dividing the new rate by the old rate. So, to find El Guerrouj's improvement over Bannister's, we divide their rates (16.134 / 15.038) to get 1.07. Percentages are found by dividing the amount of change by the old rate—(16.134 – 15.038) / 15.038 = 0.07—and multiplying the result by 100. Some people find the percentage confusing, so we use the simpler factor of improvement method. El Guerrouj was a factor-of-1.07 times faster than Bannister and about a factor-of-2 times faster than an average person.

larry's laws

"OPERATIONALLY ATTUNED"

In our daily lives, we use hundreds of devices, systems, and processes. For some of these, like the ignition on the car, we quickly learn which way to turn the key because it only turns in one direction. We don't think about how it works. Using it becomes a habit. Other gadgets are more complicated, and we have to pay attention to how they work. One example is a deadbolt lock, which moves a metal bar from the door to the doorframe so the door can't be opened. Thinking how the lock works can remind us whether the door is locked or not. Look at Figure 2.13 and notice which way the knob is turned. By picturing the inside of the lock, we can imagine that the top of the knob is attached to the bar. When the knob is pointing left, the bar must be pulled back, that is, unlocked. When the knob is positioned to the right, the bar is extended, so the door is locked. Recognizing this means that we can see at a distance whether the door is locked or unlocked. It's not a big deal, but it might save us from getting up off the sofa and trying the door to see if it's locked.

Figure 2.13 *Deadbolt lock. (a) The external view. (b) Internal components, unlocked. (c) Internal components, locked. Thinking about how the deadbolt works allows us to see at a glance whether the door is locked or not.*

EXERCISES

1. How do people learn to use technology?
2. Many computers come with a "calculator" application. Find the calculator application on your computer, and list the GUI features that help people use it.
3. The audio CD players in Figures 2.2 and 2.3 are from the same generation operating systems. Compare and contrast how the two designs help the user understand how it operates.
4. Describe the standard icons for the following operations: Cut, Copy, Paste, Print, Save, Search, and Help.
5. What are the shortcuts for the standard operations New, Open, Save, Print, Cut, Copy, Paste, Undo, and Exit?

6. Using your favorite browser as an example, list the ways it gives feedback about its operation.
7. Why is no feedback shown for any calculator operation?
8. Describe what you might do when "clicking around" a new application.
9. What is the "basic rule of information technology," as described in this chapter?
10. Find a new application on your computer or the lab's computer and "click around." List ten icons you discover, noting which are new and which you've seen before. Compare the standard operations given in Figure 2.6 with those of the application, noting which are available and any differences in shortcuts.
11. In the stoplight example in Figure 2.11, the goal was to change the color of only the red and green lights. Explain how you could do this by changing the red and green lights at the same time, but without using select-with-shift.
12. Using the application you chose in Exercise 10, spend 20 minutes "blazing away." Begin with a "blank" instance and try to use the application. Try to do as much as possible. Print your final result.
13. A basic conclusion, described as "form follows function," can be drawn from the fact that information technology is governed by the laws of nature. What is it?
14. If you are a PC user, find a Macintosh. If you are a Mac user, find a Windows PC. Explore the machine, looking under the "Start" or Apple menus. Check out the control panels. Start up your usual Web browser if it is available on the machine, or another browser if it is not. List ten differences and ten similarities that you note.
15. The basic operation of a text searcher is "match the search string to the current position in the text; if there is a match, stop and report the search string found; otherwise, move the search string one position right and repeat." How many times would that basic operation be done to find "content" in Martin Luther King's *I Have a Dream* speech?
16. Find a new word processor, possibly on the machine used in Exercise 14. Enter the following text, including all formatting as shown:

    To see the world in a grain of sand,
    And heaven in a wild flower,
    Hold infinity in the palm of your hand,
    And eternity in an hour.

    William Blake, *Auguries of Innocence*

    Use Bookman Old Style, and make your text 12 pt. with 3-inch-wide lines, and an additional half inch more overhang. Right justify the author and source. After the text, write a paragraph (Times Roman font, 10 point, normal spacing and margins) about the similarities and differences of the new word processor versus your usual word processor. Print the document.
17. What happens when we apply the $\cdot\cdot \leftarrow \cdot$ replacement to $\cdot\cdot\cdot\cdot$? Try this out on your favorite text editor.